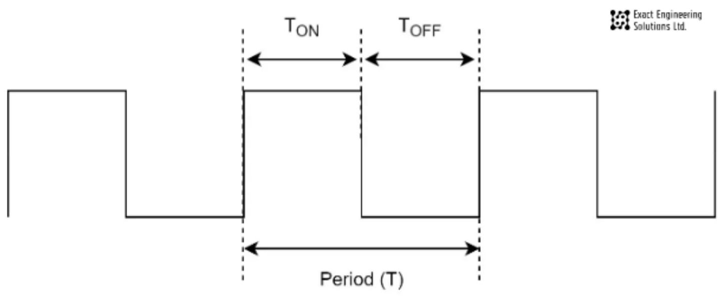
PWM Mode LED Dimmer

Pulse Width Modulation (PWM) is a technique for generating a continuous HIGH/LOW alternating digital signal and programmatically controlling its pulse width and frequency. Certain loads like (LEDs, Motors, etc) will respond to the average voltage of the signal which gets higher as the PWM signal's pulse width is increased. This technique is widely used in embedded systems to control LED brightness, motor speed, and other applications.

PWM Frequency

The PWM signal captures a few features. The first of which is the frequency, which is a measure of how fast the PWM signal keeps alternating between HIGH and LOW. The frequency is measured in Hz and it's the inverse of the full period time interval. Here is how it looks graphically and its mathematical formula.



$$Frequency[PWM] = \frac{1}{Period(T)} Hz$$

PWM Duty Cycle

The PWM's duty cycle is the most important feature that we're always interested in. It's a measure of how long the PWM signal stays ON relative to the full PWM's cycle period. The PWM's duty cycle equation is as follows:

$$DutyCycle[PWM] = \frac{T_{ON}}{T_{ON}+T_{OFF}} \times 100 = \frac{T_{ON}}{Period(T)} \times 100[\%]$$

The duty cycle is usually expressed as a percentage (%) value because it's a ratio between two-time quantities. And it directly affects the PWM's total (average) voltage that most devices respond to. That's why we typically change the duty cycle to control things like LED brightness, DC motor speed, etc.

PWM Resolution

The PWM resolution is expressed in (bits). It's the number of bits that are used to represent the duty cycle value. It can be 8bits, 10, 12, or even 16bits. The PWM resolution can be as the number of discrete duty cycle levels between 0% and 100%. The higher the PWM resolution, the higher the number of discrete levels over the entire range of the PWM's duty cycle.

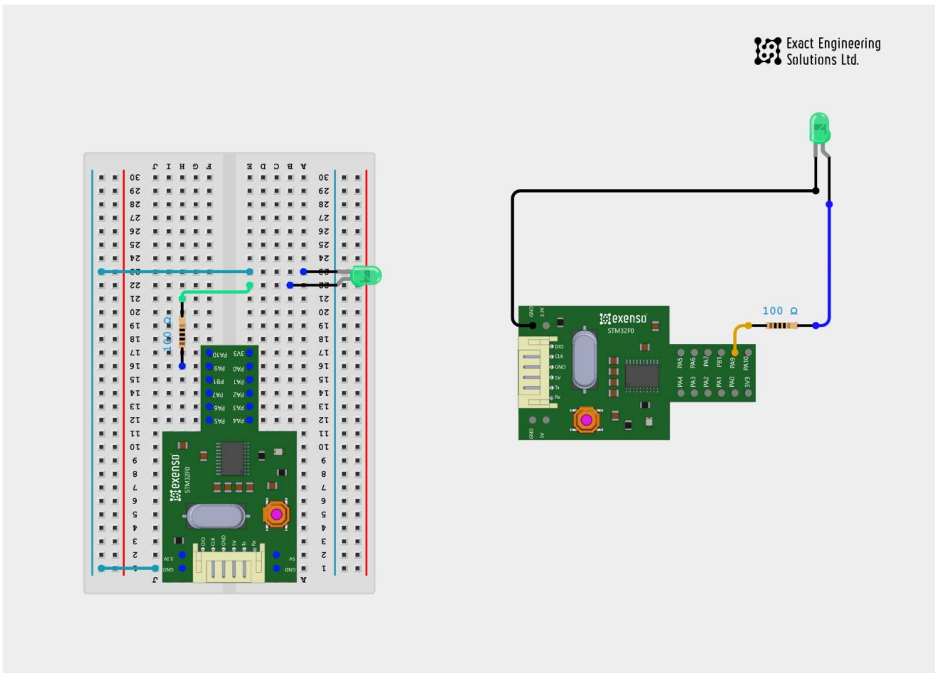
A PWM resolution of only 3 bits means there are only 8 (2^3) discrete levels for the duty cycle over the entire range (from 0% up to 100%). On the other hand, a PWM with a resolution of 8 bits will have 256(2^8) discrete levels for the duty cycle over the entire range (from 0% up to 100%).

Components Required

You will need the following components –

- 1 × Breadboard
- 1 × STM32F030F4P6
- 1 × LED
- 1 × 100Ω Resistor
- 2 × Jumper

Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.



STM32 PWM Frequency

In various applications, you'll be in need to generate a PWM signal with a specific frequency. In servo motor control, LED drivers, motor drivers, and many more situations where you'll be in need to set your desired frequency for the output PWM signal.

The PWM period ($1/F_{PWM}$) is defined by the following parameters: Auto Reload Register (ARR) value, the Prescaler value, and the internal clock itself which drives the timer module F_{CLK}. The formula down below is to be used for calculating the F_{PWM} for the output. You can set the clock you're using, the Prescaler, and solve for the ARR value to control the F_{PWM} and get what you want.

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

STM32 PWM Duty Cycle

In normal settings, assuming you're using the timer module in PWM mode and generating a PWM signal in edge-aligned mode with up-counting configuration. The duty cycle percentage is controlled by changing the value of the CCRx register. And the duty cycle equals (CCRx/ARR) [%].

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

STM32 PWM Resolution

One of the most important properties of a PWM signal is the resolution. It's the number of discrete duty cycle levels that you can set it to. This number determines how many steps the duty cycle can take until it reaches the maximum value. So, the step size or the number of duty cycle steps can tell how fine you can change the duty cycle to achieve a certain percentage. This can be extremely important in some audio applications, motor control, or even light control systems.

This is the STM32 PWM resolution formula that can be used to calculate the resolution of the PWM signal at a specific frequency or even the opposite. If you're willing to get a 10-bit resolution PWM signal, what should the frequency be to achieve this? And so on.

$$Resolution_{PWM} = \frac{log(\frac{F_{CLK}}{F_{PWM}})}{log(2)}[Bits]$$

In other situations, you'll need to adjust the ARR value. Therefore, you'll need to know the relationship between it and the PWM resolution. This is not a new formula, it's derived from the first one and the F_{PWM} equation that you've seen earlier in this tutorial.

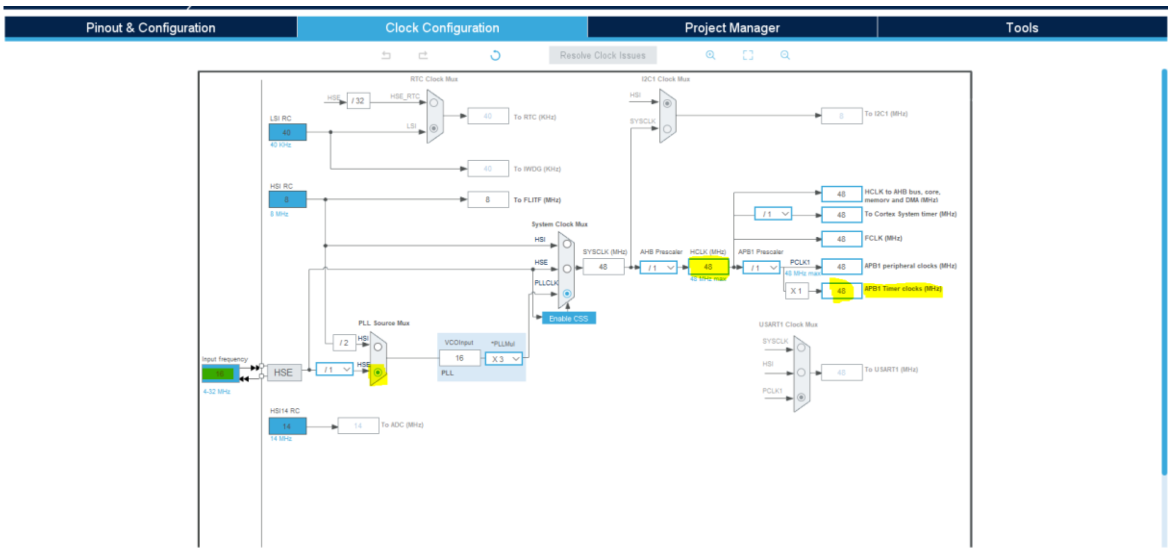
$$Resolution_{PWM}[Bits] = \frac{log(ARR + 1))}{log(2)}$$

Setup

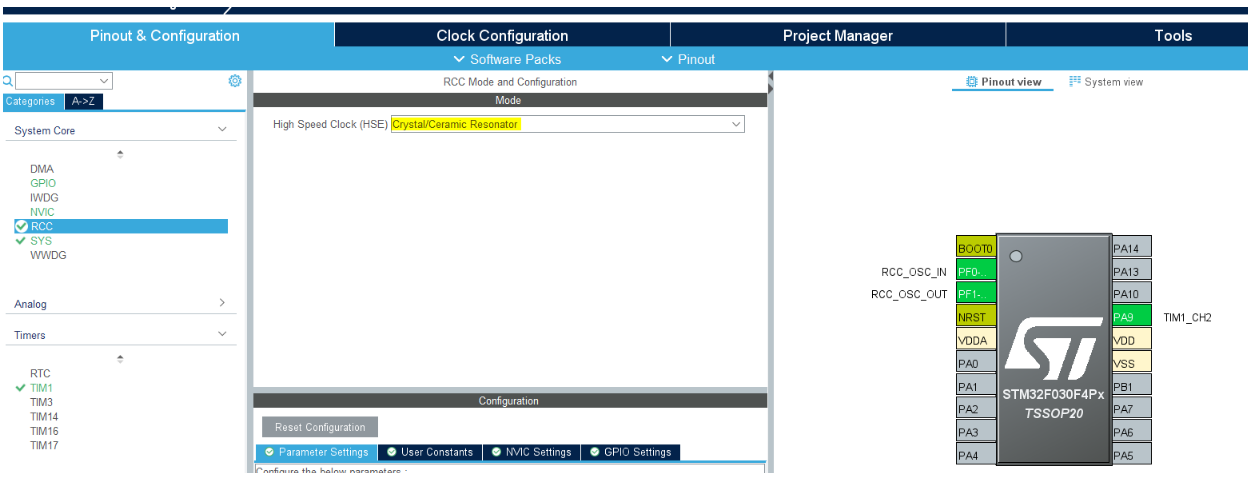
Our goal is to build a system that sweeps the duty cycle of the PWM channel 2 from 0 up to 100% back and forth. So that the LED brightness follows the same pattern. The auto-reload register will be set to a maximum value which is 65535, for no particular reason.

But you should know that the output F_{PWM} frequency is expected to be 732.433Hz from the equation we've seen earlier. And the PWM resolution is estimated to be 16-Bit which is the maximum possible value for this module.

Now, let’s start the cube IDE, and open the clock setup tab. Here you can see that, once I set the HCLK at MAX i.e. 48MHz, the APB1 clock is also at 48MHz.



This means that the TIMER1 is also running at 48MHz, as the TIM1 is connected to the APB1.



RCC Mode and Configuration

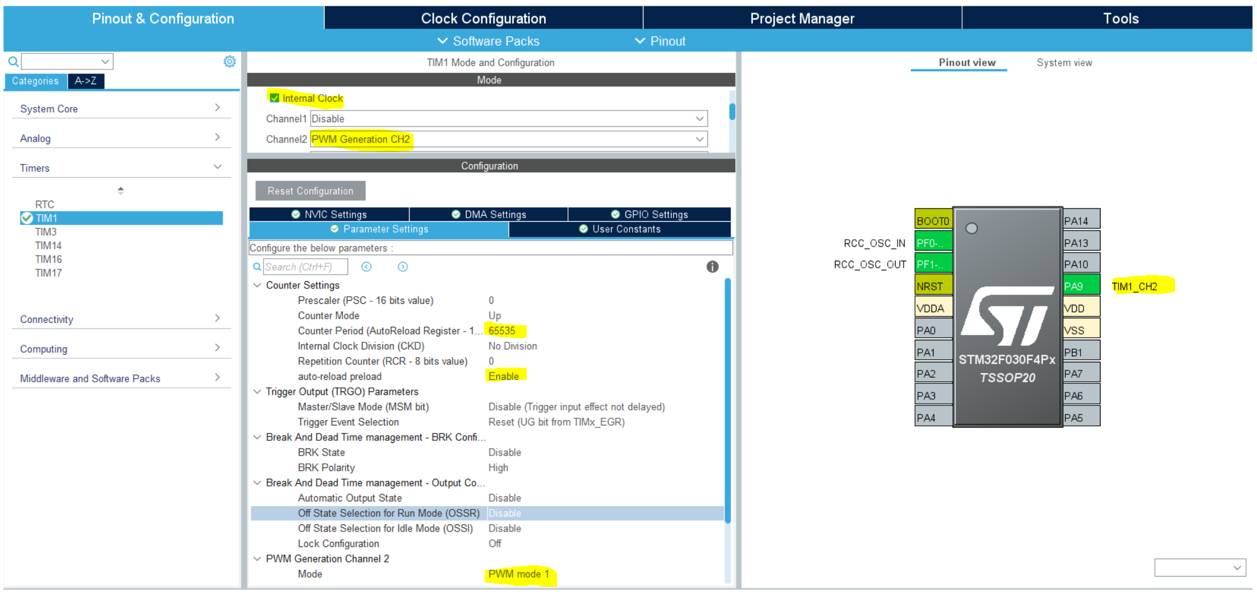
Mode

High Speed Clock (HSE) Crystal/Ceramic Resonator

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings | GPIO Settings



TIM1 Mode and Configuration

Mode

✓ Internal Clock

Channel1 Disable

Channel2 PWM Generation CH2

Configuration

Reset Configuration

NVIC Settings | DMA Settings | GPIO Settings | Parameter Settings | User Constants

Configure the below parameters :

Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 1...	65535
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
auto-reload preload	Enable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Break And Dead Time management - BRK Confi...	
BRK State	Disable
BRK Polarity	High
Break And Dead Time management - Output Co...	
Automatic Output State	Disable
Off State Selection for Run Mode (OSSR)	Disable
Off State Selection for Idle Mode (OSSI)	Disable
Lock Configuration	Off
PWM Generation Channel 2	
Mode	PWM mode 1

Code:

```c
#include "main.h"

TIM_HandleTypeDef htim1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);

int32_t CH1_DC = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM1_Init();

  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // start PWM timer1 channel 2

  while (1)
  {


      while(CH1_DC <65535)
      {
          TIM1->CCR2 = CH1_DC;
          CH1_DC += 75;
          HAL_Delay(1);
      }
      while(CH1_DC > 0)
      {
          TIM1->CCR2 = CH1_DC;
          CH1_DC -= 75;
          HAL_Delay(1);
      }

  }
}
```

You can change the line above TIMx->CCRy to match the numbers of the hardware timer you're using (x) and the PWM output channel (y).